

1. ATL Transformation Example: Ant → Maven

The Ant to Maven example describes a transformation from a file in Ant to a file in Maven (which is an extension of Ant).

1.1. Transformation overview

The aim of this transformation is to generate a file for the build tool Maven starting from a file corresponding to the build tool Ant.

```
<project name="gs-example" default="build" basedir=".">
  <target name="init">
    <tstamp/>
  </target>

  <property name="example" value="GSApp" />
  <property name="path" value="/${example}"/>
  <property name="build"
    value="${jwsdp.home}/docs/tutorial/examples/${example}/build" />
  <property name="url" value="http://localhost:8080/manager"/>
  <property file="build.properties"/>
  <property file="${user.home}/build.properties"/>

  <path id="classpath">
    <fileset dir="${jwsdp.home}/common/lib">
      <include name="*.jar"/>
    </fileset>
  </path>
  <taskdef name="install" classname="org.apache.catalina.ant.InstallTask" />
  <taskdef name="reload" classname="org.apache.catalina.ant.ReloadTask" />
  <taskdef name="remove" classname="org.apache.catalina.ant.RemoveTask"/>

  <target name="prepare" depends="init" description="Create build directories.">
    <mkdir dir="${build}" />
    <mkdir dir="${build}/WEB-INF" />
    <mkdir dir="${build}/WEB-INF/classes" />
  </target>

  <target name="install" description="Install Web application" depends="build">
    <install url="${url}" username="${username}" password="${password}"
      path="${path}" war="file:${build}"/>
  </target>

  <target name="reload" description="Reload Web application" depends="build">
    <reload url="${url}" username="${username}" password="${password}"
      path="${path}"/>
  </target>

  <target name="remove" description="Remove Web application">
    <remove url="${url}" username="${username}"
      password="${password}" path="${path}"/>
  </target>

  <target name="build" depends="prepare"
    description="Compile app Java files and copy HTML and JSP pages" >
    <javac srcdir="src" destdir="${build}/WEB-INF/classes">
      <include name="**/*.java" />
    </javac>
  </target>
</project>
```

```

    <classpath refid="classpath"/>
  </javac>
  <copy todir="${build}/WEB-INF">
    <fileset dir="web/WEB-INF" >
      <include name="web.xml" />
    </fileset>
  </copy>
  <copy todir="${build}">
    <fileset dir="web">
      <include name="*.html"/>
      <include name="*.jsp" />
      <include name="*.gif" />
    </fileset>
  </copy>
</target>
</project>

```

Figure 1. Example of file corresponding to the build tool Ant

The corresponding files in Maven are:

```

<project id="gs-example" name="gs-example">
  <build>
    <defaultGoal>build</defaultGoal>
    <sourceDirectory>.</sourceDirectory>
  </build>
</project>

```

Figure 2. project.xml

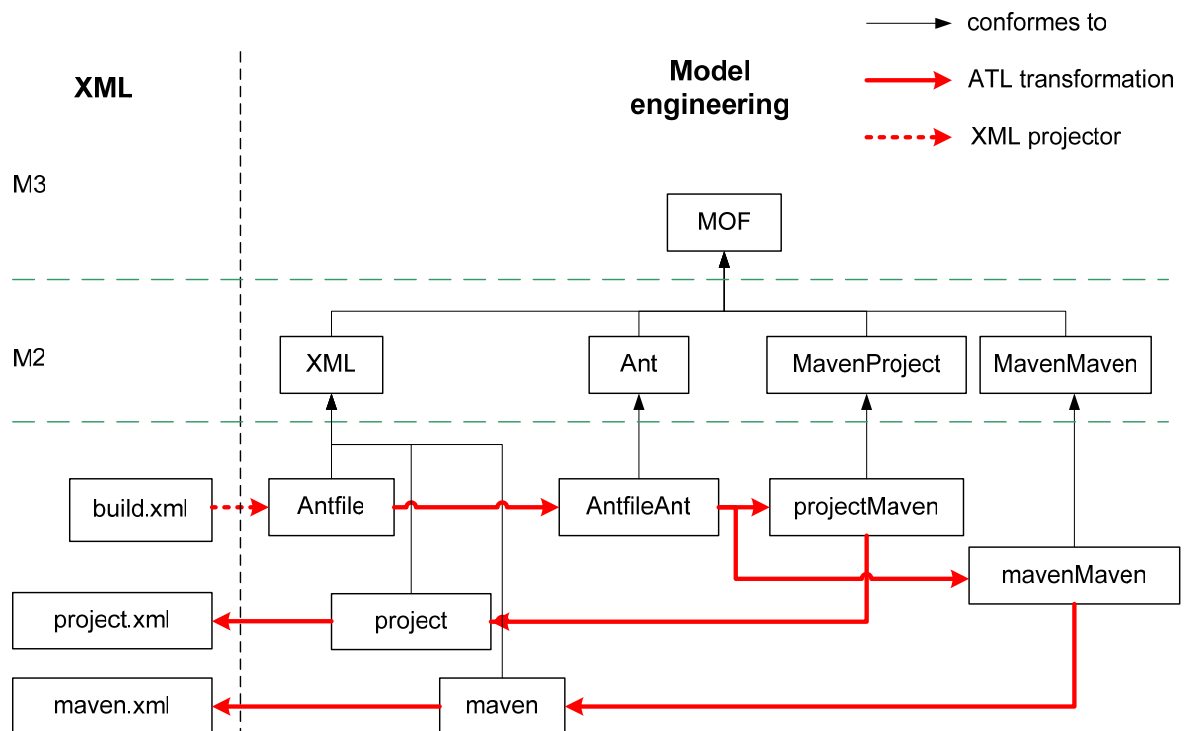
```

<project xmlns:ant="jelly:ant" default="build">
  <ant:path id="classpath">
    <ant:fileset dir="${jwsdp.home}/common/lib">
      <ant:include name="*.jar"/>
    </ant:fileset>
  </ant:path>
  <ant:property name="example" value="GSApp"/>
  <ant:property name="path" value="/${example}"/>
  <ant:property name="build"
    value="${jwsdp.home}/docs/tutorial/examples/${example}/build"/>
  <ant:property name="url" value="http://localhost:8080/manager"/>
  <ant:property file="build.properties"/>
  <ant:property file="${user.home}/build.properties"/>
  <ant:taskdef name="install" classname="org.apache.catalina.ant.InstallTask"/>
  <ant:taskdef name="reload" classname="org.apache.catalina.ant.ReloadTask"/>
  <ant:taskdef name="remove" classname="org.apache.catalina.ant.RemoveTask"/>
  <goal name="init">
    <ant:tstamp/>
  </goal>
  <goal name="prepare">
    <attainGoal name="init"/>
    <ant:mkdir ant:dir="${build}"/>
    <ant:mkdir ant:dir="${build}/WEB-INF"/>
    <ant:mkdir ant:dir="${build}/WEB-INF/classes"/>
  </goal>
  <goal name="install">
    <attainGoal name="build"/>
    <install url="${url}" username="${username}" password="${password}"
      path="${path}" war="file:${build}"/>
  </goal>

```

```

<goal name="reload">
  <attainGoal name="build"/>
  <reload url="${url}" username="${username}" password="${password}"
                                             path="${path}"/>
</goal>
<goal name="remove">
  <remove url="${url}" username="${username}" password="${password}"
                                                path="${path}"/>
</goal>
<goal name="build">
  <attainGoal name="prepare"/>
  <ant:javac srcdir="src" destdir="${build}/WEB-INF/classes">
    <ant:include name="**/*.java"/>
    <ant:classpath refid="classpath"/>
  </ant:javac>
  <ant:copy todir="${build}/WEB-INF">
    <ant:fileset dir="web/WEB-INF">
      <ant:include name="web.xml"/>
    </ant:fileset>
  </ant:copy>
  <ant:copy todir="${build}">
    <ant:fileset dir="web">
      <ant:include name="*.html"/>
      <ant:include name="*.jsp"/>
      <ant:include name="*.gif"/>
    </ant:fileset>
  </ant:copy>
</goal>
</project>
    
```

Figure 3. maven.xml

Figure 4. Transformation overview

This transformation is divided into several parts:

- the injector to obtain a file in xmi-format corresponding to the Ant Metamodel;
- the transformation from the Ant to the Maven Metamodel;
- the extractor to obtain the two files in xml-format corresponding to Maven.

1.2. Metamodels

1.2.1. Ant Metamodel

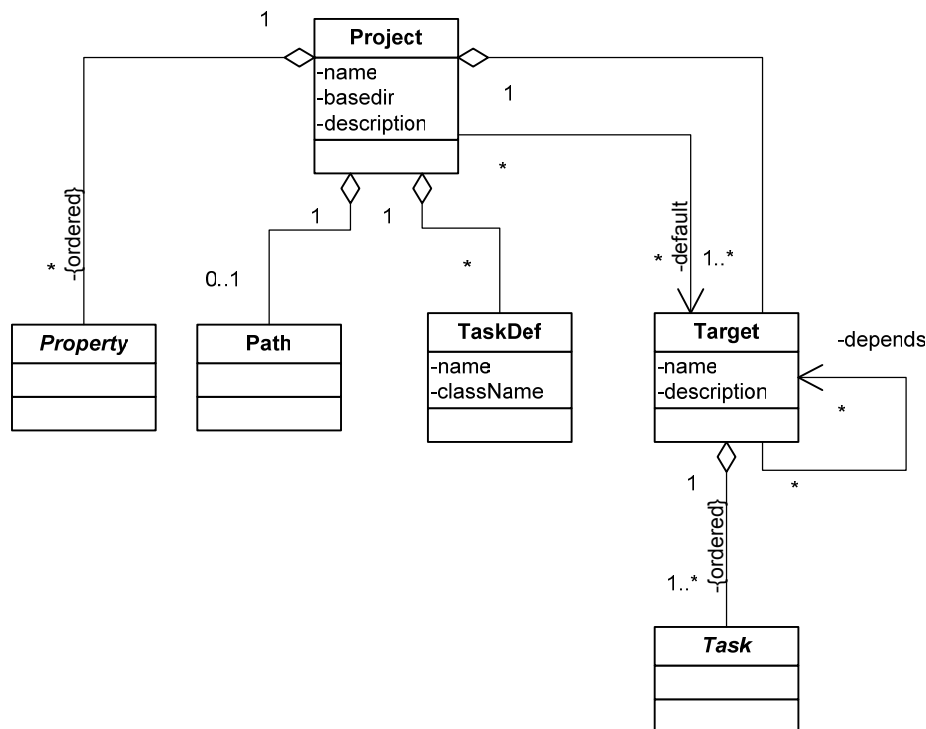


Figure 5. General Metamodel of Ant

An Ant project is modeled by a Project element. A Project element project is defined with the attributes name, basedir and description (this last attribute is optional). It contains a set of properties, a path (optional), a set of TaskDef element and at least one Target element.

A Taskdef allows adding a task definition to the current project.

A Target element is an ordered set of tasks which must be executed. It can have dependencies on other targets.

1.2.1.1. Properties

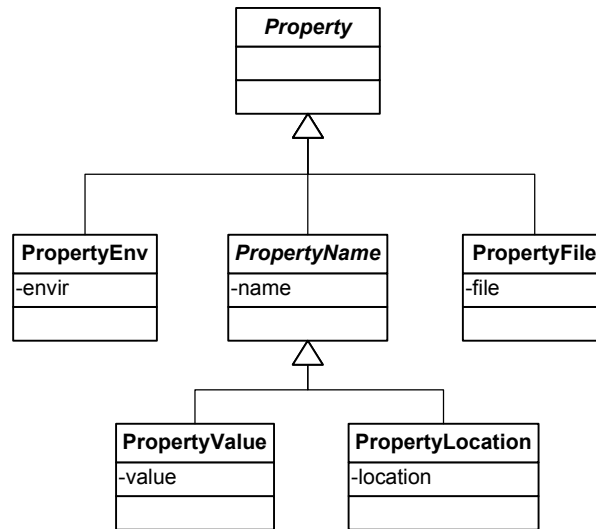


Figure 6. A few ways to define a Property

All these properties correspond to the tag 'property'.

This Metamodel allows setting various kinds of Properties:

- By supplying both the *name* and *value* attribute;
- By supplying both the *name* and *location* attribute;
- By setting the *file* attribute with the filename of the property file to load;
- By setting the *environment* attribute with a prefix to use.

1.2.1.2. Tasks

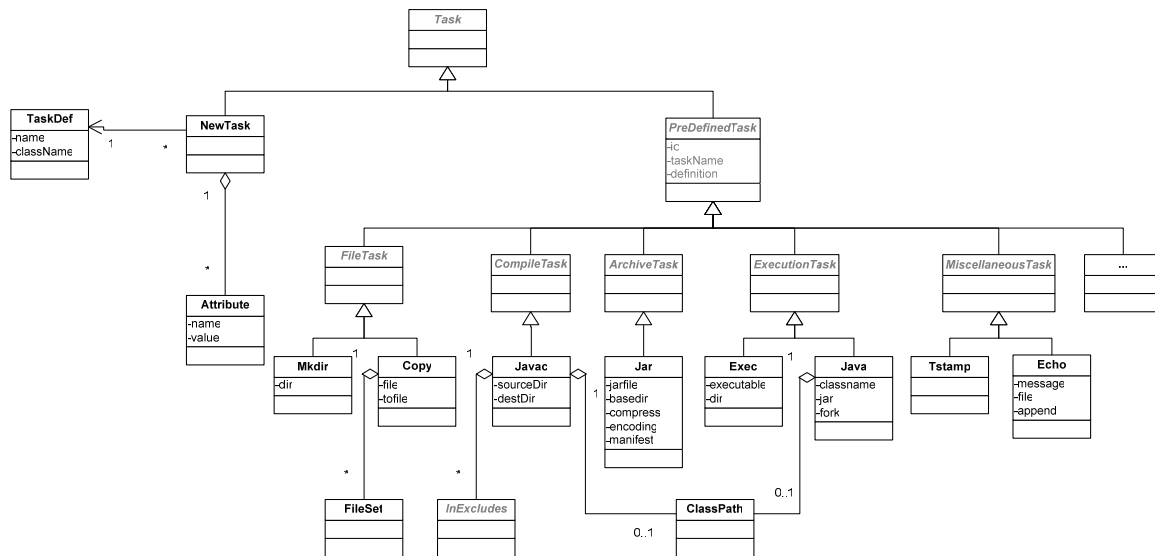


Figure 7. A few tasks

There are two types of Task:

- The tasks defined by the user. Its name is found thanks to the definition given in the TaskDef element which represents the definition of this task;
- The pre-defined tasks. There is only a sample of tasks in this Metamodel and their attributes are not all represented.

Some pre-defined tasks need a pattern (e.g. FileSet, InExcludes or ClassPath).

1.2.1.3. Pattern

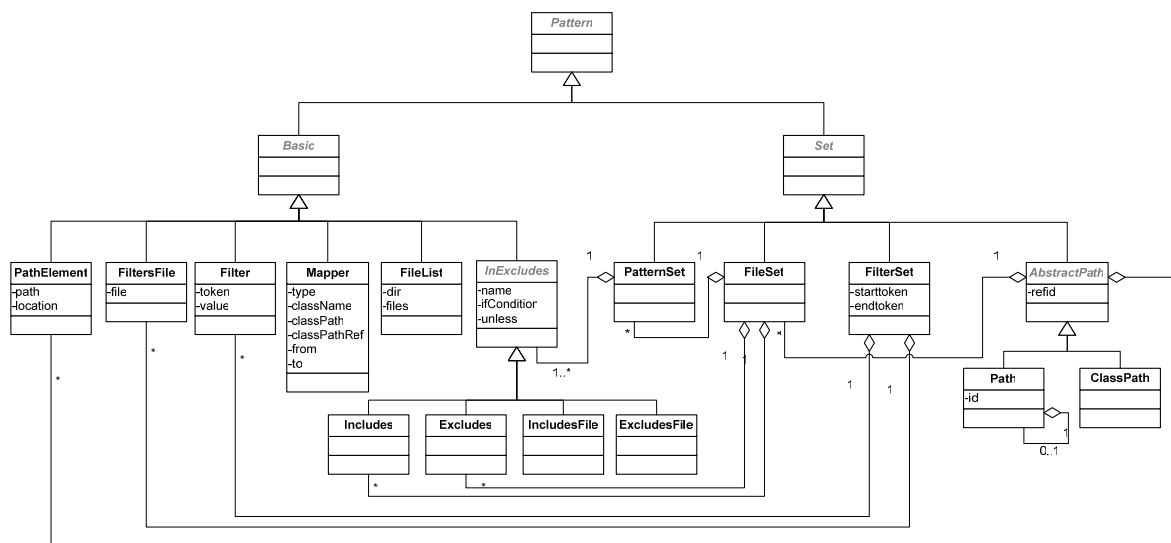



Figure 8. Metamodel of Pattern

	ATL TRANSFORMATION EXAMPLE	
	Ant to Maven	Date 05/08/2005

1.2.2. Maven Metamodels

Maven needs two XML-based files:

- project.xml, the Maven project descriptor: this file contains the basic project configuration for maven (project name, developers, urls, dependencies, etc);
- maven.xml, the Maven configuration for defining build goals: this file contains the default maven goals for the project, plus added pre-post operations to be performed.

1.2.2.1. Metamodel for the file project.xml

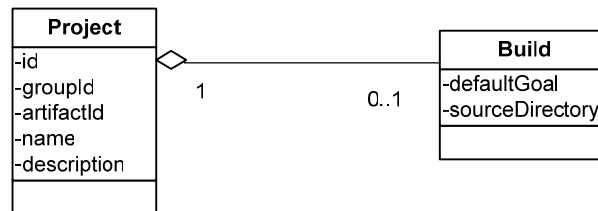
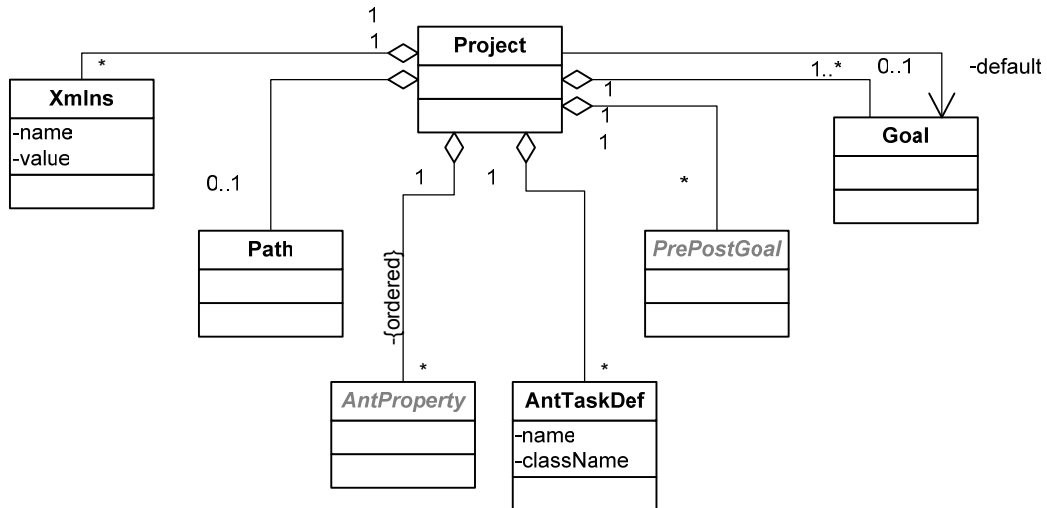


Figure 9. Metamodel of the file project.xml

A Maven project (for the file project.xml) is modeled by a Project element. A Project element is defined with the attributes id, groupId, artifactId, name, basedir description (all of these attributes are optional).

It can contain a Build element which indicates the source directory and the goal which is started by default.

It can contain others elements (like the list of developer), but these information are not deductible from an Ant file.

1.2.2.2. Metamodel for the file maven.xml

Figure 10. General Metamodel of the file maven.xml

A Maven project (for the file maven.xml) is modeled by a Project element. A Project element contains a set of Xmlns elements, an ordered set of AntProperty elements, a set of AntTaskDef elements, a set of PrePostGoal and at least one Goal element.

This project shows also the goal to start by default. But generally this information appears in the other file project.xml.

The Xmlns element represents an attribute starting with 'xmlns:' in the project tag.

The Path (and others patterns), AntProperty and AntTaskDef elements have the same definition that Path, Property and TaskDef elements in Ant presented above.

1.2.2.2.1. Goals

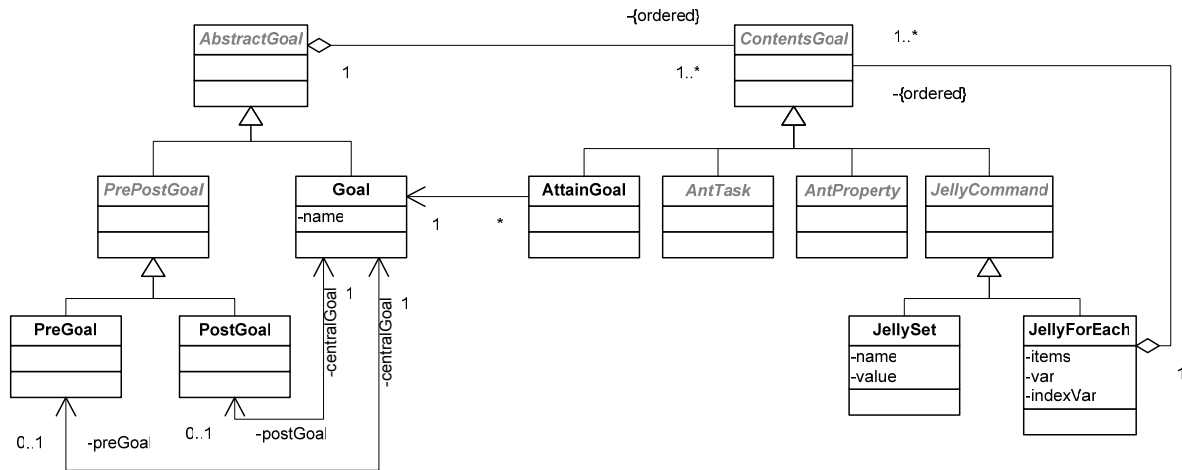


Figure 11. Goals description

An AbstractGoal element contains a list of executions.

The PreGoal element instructs Maven to execute the defined tasks in the preGoal before achieving the central goal. The PostGoal is executed after the specified goal. The PrePostGoal element is not used in this transformation.

AntTask and AntProperty elements are identical to Task and Property elements presented in Ant Metamodel.

The AttainGoal element indicates which goal must be started.


Maven can use the jelly language, represented by the JellyCommand element. The JellySet element allows giving a value to a variable. The JellyForEach element allows to make a loop. These elements are not used in this transformation.

1.3. Injector

1.3.1. Rules specification

These are the rules to transform a XML Model to an Ant Model:

- For the Root, a Project element is created,
- For an Element which name is 'target', a Target element is created,
- For an Element which name is 'property', a test on existence on its attribute must be done:
 - If this element has an attribute named 'location', a PropertyLocation element is created,
 - If this element has an attribute named 'value', a PropertyValue element is created,
 - ...
- Etc.

	ATL TRANSFORMATION EXAMPLE	
	Ant to Maven	Date 05/08/2005

1.3.2. ATL Code

This ATL code for the XML to Ant transformation consists of 7 helpers and 29 rules (one rule per element in Ant Metamodel).

The `getList` helper is useful to determine the dependencies of a target. It allows extracting a Sequence of String from a String containing words separated by a comma. This helper uses another helper named `getListAux`.

The `getAttribute` helper is useful for all elements having Attribute children. Its rule consists in returning the value of an attribute whose name is given in parameter. It returns "" if the required attribute does not exist. This helper uses `testAttribute` helper which indicates whether the attribute given in parameter exists (as children for the analysed element), and `getAttrVal` helper which returns the value of an attribute.

The `getElement` helper is useful for all elements having Text children. Its rule consists in returning the value of a Text element contained in an Element whose name is given in parameter. It returns "" if the required element does not exist. This helper uses `testElement` helper which indicates whether the Element given in parameter exists (as children for the analysed element).

The rule `Root2Project` allocates a Project element.

The rule `Target` allocates a Target element.

...

For the rule `Root2Project`, the reference 'default' needs an Element named 'target' whose value of the Attribute named 'name' has the same value as that given in the Attribute of name 'default':

```
default <- XML!Element.allInstances() ->
  select(d | d.name = 'target'
    and d.getAttribute('name')=i.getAttribute('default')) ->
    first(),
```

For the rule `Target`, the reference 'depends' needs all Element named 'target' whose value of the Attribute named 'name' is included in the list containing the dependencies.

```
depends <- XML!Element.allInstances() ->
  select(d | d.name = 'target'
    and thisModule.getList(i.getAttribute('depends'))->
      includes(d.getAttribute('name'))),
```

Concerning the rule `NewTask`, a test is done on the existence of this new Task, that is to say that an Element named 'taskdef' must have the same value (in the Attribute named 'name') as the name of this Element. To find the reference for `taskName`, a research on all the elements and a selection on the name are done.

```
rule NewTask{
  from i : XML!Element(
    -- this task must be defined
    not(XML!Element.allInstances() ->
      select(d | d.name = 'taskdef'
        and d.getAttribute('name')=i.name) ->
        isEmpty())
  )
  to o : Ant!NewTask(
    -- reference to the definition of this task
    taskName <- XML!Element.allInstances() ->
      select(d | d.name = 'taskdef'
        and d.getAttribute('name')=i.name) ->
```

```
        first(),
    -- its attributes
    attributes <- i.children ->
        select(d | d.ocIsKindOf(XML!Attribute))
    )
}
```

Concerning the rule `Attribut`, a test is done on the existence of this new Task on the parent, that is to say that an Element named 'taskdef' must have the same value (in the Attribute named 'name') as the name of the parent of this Element.

```
rule Attribut{
  from i : XML!Attribute(
    not(XML!Element.allInstances() ->
      select(d | d.name = 'taskdef'
        and d.getAttribute('name')=i.parent.name) ->
      isEmpty())
  )
  to o : Ant!Attribut(
    name <- i.name,
    value<- i.value
  )
}
```

1.4. Transformation from Ant to Maven

This transformation has one file in entry corresponding to the Ant Metamodel and it creates two files: one corresponds to the MavenMaven Metamodel (which represents the file `maven.xml`) and the other corresponds to the MavenProject Metamodel (which represents the file `project.xml`).

1.4.1. Rules Specification

These are the rules to transform an Ant model to Maven model:


- For a Project element, a Project and Build elements for the file `maven.xml` (MavenMaven Metamodel) are created, and a Project element for the file `project.xml` (MavenProject Metamodel) is created.
- For a Target element, a Goal element is created.
- For all properties, tasks and pattern, the elements are simply copied: for a PropertyValue element, an AntPropertyValue element is created, etc.

1.4.2. ATL Code

This ATL code for the Ant to Maven transformation consists of 30 rules.

The rule `AntProject2Maven` allocates a Project, Build and Xmlns elements of the MavenMaven Metamodel and a Project element of the MavenProject Metamodel. There are two kinds of Project: those which have a description and those which do not have (in this case, the rule `AntProject2MavenWithoutDescription` is started).

The rule `AntTarget2MavenMavenGoal` allocates a Goal element (of the MavenMaven Metamodel). In the reference contentsGoal, the dependencies of a target appear before its tasks. For each dependency, an `AttainGoal` element is created. Those are separately treated thanks to the use of 'distinct foreach'.

	ATL TRANSFORMATION EXAMPLE	
	Ant to Maven	Date 05/08/2005

```

rule AntTarget2MavenMavenGoal{
  from a : Ant!Target
  using {
    itsDependencies : Sequence(Ant!Target) = a.depends->asSequence();
  }
  to mg : MavenMaven!Goal(
    name <- a.name,
    contentsGoal <- Sequence{dependencies,a.tasks}
  ),
  -- for all element g in the Sequence itsDependencies
  dependencies : distinct MavenMaven!AttainGoal foreach(g in itsDependencies) (
    attainGoal <- g
  )
}

```

All the others rules are simple copies of property, task or pattern.

1.5. Extractor

It creates two files corresponding to XML Metamodel from two files: one corresponding to the MavenMaven Metamodel ant the other corresponding to the MavenProject Metamodel. The files maven.xml and project.xml are together used in Maven, that is why their transformation (which are independent each other) appears in the same file.

1.5.1. Rules specification

These are the rules to transform a MavenMaven and MavenProject Model to 2 XML Models:

- For the Project corresponding to MavenMaven Metamodel, a Root element is created (for the XMLMaven Metamodel),
- For the elements existing in Ant like Property, an Element is created and it contains an Attribute with the name 'name' and the value 'ant:property' because in the Xmlns element having the attribute 'jelly:ant', the value is 'xmlns:ant', that is to say that all elements existing in Ant which are called begins by 'ant:'. If the value of this Xmlns element is 'xmlns', that is to say that elements existing in Ant do not need a prefix;
- For the Project corresponding to MavenProject Metamodel, a Root element is created (for the XMLProject Metamodel),
- ...

1.5.2. ATL Code

This ATL code for the Maven to XML transformation consists of 2 helpers 25 rules concerning the MavenMaven Metamodel and 4 rules concerning the MavenProject Metamodel.


The getXmlns helper returns the prefix used to call an execution which in a tag library (e.g. jelly:ant). It uses the getXmlnsAux helper: it allows returning the name of the Xmlns element having the same value as that given in parameter.

This helper is used for an AntPropertyValue element to determine the name of the element:

```

rule PropertyValue{

```

	ATL TRANSFORMATION EXAMPLE	
	Ant to Maven	Date 05/08/2005

```
from i : MavenMaven!AntPropertyValue
to o : XMLMaven!Element(
  name <- thisModule.getXmlns('jelly:ant')+'property',
  children <- Sequence{propertyName2,propertyValue}
),
...}
```

There is a rule for each element.

I. Ant Metamodel in KM3

```
1 package Ant{
2   -- @begin central element
3   class Project{
4     attribute name [0-1] : String;
5     attribute basedir [0-1] : String;
6     attribute description [0-1] : String;
7     reference "default" : Target;
8     reference path [0-1] container : Path;
9     reference properties [*] ordered container : Property;
10    reference taskdef [*] container : TaskDef;
11    reference targets [1-*] ordered container : Target;
12  }
13  -- @end central element
14
15
16  -- @begin property
17  -- @comments represents the properties for a project
18  abstract class Property {}
19
20  class PropertyName extends Property{
21    attribute name : String;
22  }
23
24  -- @comments represents a property to set a value
25  class PropertyValue extends PropertyName{
26    attribute value : String;
27  }
28
29  -- @comments represents a property set to the absolute filename
30  -- of the given file
31  class PropertyLocation extends PropertyName{
32    attribute location : String;
33  }
34
35  -- @comments represents a property file to load
36  class PropertyFile extends Property{
37    attribute file : String;
38  }
39
40  -- @comments represents a property retrieving environment variables
41  class PropertyEnv extends Property{
42    attribute environment : String;
43  }
44  -- @end property
45
46
47  -- @begin target
48  -- @comments represents a set of tasks which must be executed
49  class Target{
50    attribute name : String;
51    attribute description[0-1] : String;
52    attribute unless [0-1] : String;
53    attribute ifCondition [0-1] : String;
54    reference depends [*] : Target;
55    reference tasks [*] ordered container : Task oppositeOf target;
56  }
```

```
57  -- @end target
58
59
60  -- @begin pattern
61  -- @comments represents complex parameters for some tasks
62  abstract class Pattern{}
63
64  -- @begin basicPattern
65  -- @comments represents a basic parameter (no children)
66  abstract class Basic extends Pattern{}
67
68  -- @comments represents the tag 'mapper' (mapping file names)
69  class Mapper extends Basic{
70      attribute type [0-1] : String;
71      attribute classname [0-1] : String;
72      attribute classpath [0-1] : String;
73      attribute classpathref [0-1] : String;
74      attribute from [0-1] : String;
75      attribute to [0-1] : String;
76  }
77
78  -- @comments represents the tag 'include','exclude',
79  -- 'includeFile' and 'excludeFile'(including or excluding files)
80  abstract class InExcludes extends Basic{
81      attribute name : String;
82      attribute ifCondition [0-1] : String;
83      attribute unless [0-1] : String;
84  }
85
86  class Includes extends InExcludes{}
87  class Excludes extends InExcludes{}
88  class IncludesFile extends InExcludes{}
89  class ExcludesFile extends InExcludes{}
90
91  -- @comments represents lists of files
92  class FileList extends Basic{
93      attribute dir : String;
94      attribute files : String;
95  }
96
97  -- @comments represents a filter : to replace a token value
98  class Filter extends Basic{
99      attribute token : String;
100     attribute value : String;
101 }
102
103 -- @comments represents the tag filtersfile:
104 -- to load a file containing name value pairs
105 class FiltersFile extends Basic{
106     attribute file : String;
107 }
108
109 -- @comments represents the tag pathelement
110 class PathElement extends Basic{
111     attribute path : String;
112     attribute location : String;
113 }
114 -- @end basicPattern
115 -- @begin setPattern
116 -- @comments represents set parameters
117 abstract class Set extends Pattern{}
118
```


```
119 -- @comments represents the tag 'patternset'
120 class PatternSet extends Set{
121     reference inExcludes [1-*] container : InExcludes;
122 }
123
124 -- @comments represents the tag 'fileset' representing a group of files
125 class FileSet extends Set{
126     attribute dir : String;
127     reference patternset [*] container : PatternSet;
128     reference include [*] container : Includes;
129     reference exclude [*] container : Excludes;
130 }
131
132 -- @comments represents the tag 'filterset'
133 -- representing a group of filters
134 class FilterSet extends Set{
135     attribute startToken [0-1] : String;
136     attribute endToken [0-1] : String;
137     reference filter [*] container : Filter;
138     reference filtersFile [*] container : FiltersFile;
139 }
140
141 abstract class AbstractPath extends Set{
142     attribute refid [0-1] : String;
143     reference pathElement [*] container : PathElement;
144     reference fileset [*] container : FileSet;
145 }
146
147 -- @comments represents the tag 'path'
148 class Path extends AbstractPath{
149     attribute id : String;
150     reference path [0-1] container : Path;
151 }
152
153 -- @comments represents the tag 'classpath'
154 class ClassPath extends AbstractPath{
155 }
156 -- @begin setPattern
157 -- @end pattern
158
159 -- @begin task
160 -- @comments represents a piece of code
161 abstract class Task{
162     reference target : Target oppositeOf tasks;
163 }
164 -- @begin newTask
165 -- @comments represents a task defined by the user
166 class TaskDef{
167     attribute name : String;
168     attribute classname : String;
169 }
170
171 -- @comments represents a call of a task created by the user
172 class NewTask extends Task {
173     reference taskName : TaskDef;
174     reference attributes[*] container : Attribut;
175 }
176
177 -- @comments represents a attribute used in a new task
178 class Attribut{
179     attribute name : String;
180     attribute value : String;
```



```
181     }
182     -- @end newTask
183
184     -- @begin predefinedTasks
185     -- @comments represents predefined tasks
186     abstract class PreDefinedTask extends Task{
187         attribute id [0-1] : String;
188         attribute taskname [0-1] : String;
189         attribute description [0-1] : String;
190     }
191
192     -- @begin executionTasks
193     abstract class ExecutionTask extends PreDefinedTask{}
194
195     -- @comments represents the tag 'exec': execute a system command
196     class Exec extends ExecutionTask{
197         attribute executable : String;
198         attribute dir : String;
199     }
200
201     -- @comments represents the tag 'java': execute a java class
202     class Java extends ExecutionTask{
203         attribute classname : String;
204         attribute jar [0-1] : String;
205         attribute fork [0-1] : String;
206         reference classPath [0-1] container : ClassPath;
207     }
208     -- @end executionTasks
209
210
211     -- @begin miscellaneousTasks
212     abstract class MiscellaneousTask extends PreDefinedTask{}
213
214     -- @comments represents the tag 'echo':
215     -- echoes text to System.out or to a file
216     class Echo extends MiscellaneousTask{
217         attribute message : String;
218         attribute file [0-1] : String;
219         attribute append [0-1] : String;
220     }
221
222     -- @comments represents the tag 'tstamp': set the tstamp
223     class Tstamp extends MiscellaneousTask{
224         reference format[*] container : FormatTstamp;
225     }
226
227     class FormatTstamp{
228         attribute property : String;
229         attribute pattern : String;
230         attribute offset [0-1] : String;
231         attribute unit [0-1] : String;
232         attribute locale [0-1] : String;
233     }
234     -- @end miscellaneousTasks
235
236     -- @begin compileTasks
237     abstract class CompileTask extends PreDefinedTask{}
238
239     -- @comments represents the tag 'javac':
240     -- compiles the specified source file(s)
241     class Javac extends CompileTask{
242         attribute srcdir : String;
```

```
243     attribute destdir [0-1]: String;
244     attribute debug [0-1] : String;
245     attribute fork [0-1] : String;
246     attribute optimize [0-1] : String;
247     attribute deprecation [0-1] : String;
248     reference inExcludes[*] container : InExcludes;
249     reference classPath [0-1] container : ClassPath;
250 }
251 -- @end compileTasks
252
253 -- @begin documentationTasks
254 abstract class DocumentationTask extends PreDefinedTask{}
255
256 class Javadoc extends DocumentationTask{
257     attribute sourcepath : String;
258     attribute destdir : String;
259     attribute packagenames : String;
260     attribute defaultexcludes : String;
261     attribute author : String;
262     attribute version : String;
263     attribute use : String;
264     attribute windowtitle : String;
265 }
266 -- @end documentationTasks
267
268 -- @begin archiveTasks
269 abstract class ArchiveTask extends PreDefinedTask{}
270
271 -- @comments represents the tag 'jar': jars a set of files
272 class Jar extends ArchiveTask{
273     attribute jarfile : String;
274     attribute basedir [0-1] : String;
275     attribute compress [0-1] : String;
276     attribute encoding [0-1] : String;
277     attribute manifest [0-1] : String;
278 }
279 -- @end archiveTasks
280
281 -- @begin fileTasks
282 abstract class FileTask extends PreDefinedTask{}
283
284 -- @comments represents the tag 'mkdir': creates a directory
285 class Mkdir extends FileTask{
286     attribute dir : String;
287 }
288
289 -- @comments represents the tag 'copy':
290 -- copies a file or Fileset to a new file or directory
291 class Copy extends FileTask{
292     attribute file [0-1] : String;
293     attribute presserelastmodified [0-1] : String;
294     attribute tofile [0-1] : String;
295     attribute todir [0-1] : String;
296     attribute overwrite [0-1] : String;
297     attribute filtering [0-1] : String;
298     attribute flatten [0-1] : String;
299     attribute includeEmptyDirs [0-1] : String;
300     reference fileset [0-1] container : FileSet;
301     reference filterset [0-1] container : FilterSet;
302     reference mapper [0-1] container : Mapper;
303 }
304
```

```
305 -- @comments represents the tag 'delete':
306 -- deletes either a single file,
307 -- all files and sub-directories in a specified directory,
308 -- or a set of files specified by one or more FileSets
309 class Delete extends FileTask{
310     attribute file [0-1] : String;
311     attribute dir [0-1] : String;
312     attribute verbose [0-1] : String;
313     attribute quiet [0-1] : String;
314     attribute failonerror [0-1] : String;
315     attribute includeEmptyDirs [0-1] : String;
316     attribute includes [0-1] : String;
317     attribute includesfile [0-1] : String;
318     attribute excludes [0-1] : String;
319     attribute excludesfile [0-1] : String;
320     attribute defaultexcludes [0-1] : String;
321 }
322 -- @end fileTasks
323
324 -- @begin executionTasks
325 abstract class ExecutionTask extends PreDefinedTask{}
326
327 -- @comments represents the tag 'exec': executes a system command
328 class Exec extends ExecutionTask{
329     attribute executable : String;
330     attribute dir : String;
331 }
332 -- @end executionTasks
333 -- @end task
334 }
335
336 package PrimitiveTypes{
337     datatype String;
338 }
```

	ATL TRANSFORMATION EXAMPLE	
	Ant to Maven	Date 05/08/2005

II. Maven Metamodel in KM3

II.1 Project.xml

```

1  package MavenProject {
2
3     -- @comments represents the current project
4     class Project{
5         attribute id [0-1] : String;
6         attribute groupId [0-1] : String;
7         attribute artifactId [0-1] : String;
8         attribute name [0-1] : String;
9         attribute description [0-1] : String;
10        reference build [0-1] container : Build;
11    }
12
13    -- @comments represents the tag 'build'
14    -- containing the informations required to build the project
15    class Build{
16        attribute defaultGoal [0-1] : String;
17        attribute sourceDirectory : String;
18        attribute unitTestSourceDirectory [0-1] : String;
19        reference uniTest [*] : Resource;
20        reference resources [*] : Resource;
21    }
22 }
23 package PrimitiveTypes{
24     datatype String;
25 }

```

II.2 Maven.xml

```
1 package MavenMaven {
2   -- @begin project
3   -- @comments central element of the file
4   class Project {
5     reference xmlns [*] container : Xmlns;
6     reference "default" [0-1] : Goal;
7     reference path [0-1] container : Path;
8     reference properties [*] ordered container : AntProperty;
9     reference taskdefs [*] container : AntTaskDef;
10    reference prePostGoals [*] container : PrePostGoal;
11    reference goals [1-*] container : Goal;
12  }
13  -- @end project
14
15  class Xmlns {
16    attribute name: String;
17    attribute value : String;
18  }
19
20  -- @begin antProperty
21  -- @comments represents the tag 'property': the properties for a project
22  abstract class AntProperty extends ContentsGoal{}
23
24  abstract class AntPropertyName extends AntProperty{
25    attribute name : String;
26  }
27  -- @comments represents a property to set a value
28  class AntPropertyValue extends AntPropertyName{
29    attribute value : String;
30  }
31  -- @comments represents a property set
32  --to the absolute filename of the given file
33  class AntPropertyLocation extends AntPropertyName{
34    attribute location : String;
35  }
36  -- @comments represents a property file to load
37  class AntPropertyFile extends AntProperty{
38    attribute file : String;
39  }
40  -- @comments represents a property retrieving environment variables
41  class AntPropertyEnv extends AntProperty{
42    attribute environment : String;
43  }
44  -- @end antProperty
45
46  -- @begin jellyCommands
47  abstract class JellyCommand extends ContentsGoal{}
48
49  -- @comments The set tag sets the jelly variable named by the var
50  -- attribute to the value given by the value attribute.
51  -- @comments Unlike Ant properties, Jelly variables can be changed
52  -- once they have been given a value
53  class JellySet extends JellyCommand{
54    attribute var : String;
55    attribute value : String;
56  }
```

```
57
58 class JellyForEach extends JellyCommand{
59     attribute items : String;
60     attribute var : String;
61     attribute indexVar : String;
62     reference contents ordered container : ContentsGoal;
63 }
64 -- @end jellyCommands
65
66 -- @begin goals
67 -- @comments represents a set of tasks which must be executed
68 abstract class AbstractGoal{
69     reference contentsGoal [1-*] ordered container : ContentsGoal;
70 }
71
72 abstract class ContentsGoal{}
73
74 class AttainGoal extends ContentsGoal{
75     reference attainGoal : Goal;
76 }
77
78 -- @comments represent extensions of a goal
79 abstract class PrePostGoal extends AbstractGoal{}
80
81 class PreGoal extends PrePostGoal{
82     reference centralGoal : Goal oppositeOf preGoal;
83 }
84
85 class PostGoal extends PrePostGoal{
86     reference centralGoal : Goal oppositeOf postGoal;
87 }
88
89 -- @comments represents a goal
90 class Goal extends AbstractGoal{
91     attribute name : String;
92     reference preGoal [0-1] : PreGoal oppositeOf centralGoal;
93     reference postGoal [0-1] : PostGoal oppositeOf centralGoal;
94 }
95 -- @end goals
96
97 -- @begin pattern
98 -- @comments represents complex parameters for some tasks
99 abstract class Pattern{}
100
101 -- @begin basicPattern
102 -- @comments represents a basic parameter(no children)
103 abstract class Basic extends Pattern{}
104
105 -- @comments represents the tag 'mapper' (mapping file names)
106 class Mapper extends Basic{
107     attribute type [0-1] : String;
108     attribute classname [0-1] : String;
109     attribute classpath [0-1] : String;
110     attribute classpathref [0-1] : String;
111     attribute from [0-1] : String;
112     attribute to [0-1] : String;
113 }
114
115 -- @comments represents the tag 'include','exclude',
116 -- 'includeFile' and 'excludeFile'(including or excluding files)
117 abstract class InExcludes extends Basic{
118     attribute name : String;
```


```
119     attribute ifCondition [0-1] : String;
120     attribute unless [0-1] : String;
121 }
122
123 class Includes extends InExcludes{}
124 class Excludes extends InExcludes{}
125 class IncludesFile extends InExcludes{}
126 class ExcludesFile extends InExcludes{}
127
128 -- @comments represents lists of files
129 class FileList extends Basic{
130     attribute dir : String;
131     attribute files : String;
132 }
133
134 -- @comments represents a filter: to replace a token value
135 class Filter extends Basic{
136     attribute token : String;
137     attribute value : String;
138 }
139
140 -- @comments represents the tag filtersfile:
141 -- to load a file containing name value pairs
142 class FiltersFile extends Basic{
143     attribute file : String;
144 }
145
146 -- @comments represents the tag 'pathelement'
147 class PathElement extends Basic{
148     attribute path : String;
149     attribute location : String;
150 }
151 -- @end basicPattern
152
153 -- @begin setPattern
154 -- @comments represents set parameters
155 abstract class Set extends Pattern{}
156
157 -- @comments represents the tag 'patternset'
158 class PatternSet extends Set{
159     reference inexcludes [1-*] container : InExcludes;
160 }
161
162 -- @comments represents the tag 'fileset' representing a group of files
163 class FileSet extends Set{
164     attribute dir : String;
165     reference patternset [*] container : PatternSet;
166     reference include [*] container : Includes;
167     reference exclude [*] container : Excludes;
168 }
169
170 -- @comments represents the tag 'filterset'
171 -- representing a group of filters
172 class FilterSet extends Set{
173     attribute starttoken [0-1] : String;
174     attribute endtoken [0-1] : String;
175     reference filter [*] container : Filter;
176     reference filtersfile [*] container : FiltersFile;
177 }
178
179 -- @comments represents the tag 'path'
180 class Path extends Set{
```

```
181     attribute id : String;
182     attribute refid [0-1] : String;
183     reference path [0-1] container : Path;
184     reference pathElement [*] container : PathElement;
185     reference fileset [*] container : FileSet;
186 }
187
188 -- @comments represents the tag 'classpath'
189 class ClassPath extends Set{
190     attribute refid : String;
191     reference pathElement [*] container : PathElement;
192     reference fileset [*] container : FileSet;
193 }
194 -- @end setPattern
195 -- @end pattern
196
197 -- @begin antTasks
198 -- @comments represents a piece of code
199 abstract class Task extends ContentsGoal{}
200
201 -- @begin newTask
202 -- @comments represents a task defined by the user
203 class AntTaskDef extends ContentsGoal{
204     attribute name : String;
205     attribute classname : String;
206 }
207
208 -- @comments represents a call of a task created by the user
209 class NewTask extends Task {
210     reference taskName : AntTaskDef;
211     reference attributes[*] container : Attribut;
212 }
213
214 -- @comments represents a attribute used in a new task
215 class Attribut{
216     attribute name : String;
217     attribute value : String;
218 }
219 -- @end newTask
220
221 -- @begin predefinedTasks
222 -- @comments represents predefined tasks
223 abstract class PreDefinedTask extends Task{
224     attribute id [0-1] : String;
225     attribute taskname [0-1] : String;
226     attribute description [0-1] : String;
227 }
228
229 -- @begin executionTasks
230 abstract class ExecutionTask extends PreDefinedTask{}
231
232 -- @comments represents the tag 'exec': execute a system command
233 class Exec extends ExecutionTask{
234     attribute executable : String;
235     attribute dir : String;
236 }
237
238 -- @comments represents the tag 'java': execute a java class
239 class Java extends ExecutionTask{
240     attribute classname : String;
241     attribute jar [0-1] : String;
242     attribute fork [0-1] : String;
```



```
243     reference classPath [0-1] container : ClassPath;
244 }
245 -- @end executionTasks
246
247 -- @begin miscellaneousTasks
248 abstract class MiscellaneousTask extends PreDefinedTask{}
249
250 -- @comments represents the tag 'echo':
251 -- echoes text to System.out or to a file
252 class Echo extends MiscellaneousTask{
253     attribute message : String;
254     attribute file [0-1] : String;
255     attribute append [0-1] : String;
256 }
257
258 -- @comments represents the tag 'tstamp' : set the tstamp
259 class Tstamp extends MiscellaneousTask{
260     reference format[*] container : FormatTstamp;
261 }
262
263 class FormatTstamp{
264     attribute property : String;
265     attribute pattern : String;
266     attribute offset [0-1] : String;
267     attribute unit [0-1] : String;
268     attribute locale [0-1] : String;
269 }
270 -- @end miscellaneousTasks
271
272 -- @begin compileTasks
273 abstract class CompileTask extends PreDefinedTask{}
274
275 -- @comments represents the tag 'javac':
276 -- compiles the specified source file(s)
277 class Javac extends CompileTask{
278     attribute srcdir : String;
279     attribute destdir [0-1]: String;
280     attribute debug [0-1] : String;
281     attribute fork [0-1] : String;
282     attribute optimize [0-1] : String;
283     attribute deprecation [0-1] : String;
284     reference inExcludes[*] container : InExcludes;
285     reference classPath [0-1] container : ClassPath;
286 }
287 -- @end compileTasks
288
289 -- @begin documentationTasks
290 abstract class DocumentationTask extends PreDefinedTask{}
291
292 class Javadoc extends DocumentationTask{
293     attribute sourcepath : String;
294     attribute destdir : String;
295     attribute packagenames : String;
296     attribute defaultexcludes : String;
297     attribute author : String;
298     attribute version : String;
299     attribute use : String;
300     attribute windowtitle : String;
301 }
302 -- @end documentationTasks
303
304 -- @begin archiveTasks
```

```
305     abstract class ArchiveTask extends PreDefinedTask{}
306
307     -- @comments represents the tag 'jar': jars a set of files
308     class Jar extends ArchiveTask{
309         attribute jarfile : String;
310         attribute basedir [0-1] : String;
311         attribute compress [0-1] : String;
312         attribute encoding [0-1] : String;
313         attribute manifest [0-1] : String;
314     }
315     -- @end archiveTasks
316
317     -- @begin fileTasks
318     abstract class FileTask extends PreDefinedTask{}
319
320     -- @comments represents the tag 'mkdir': creates a directory
321     class Mkdir extends FileTask{
322         attribute dir : String;
323     }
324
325     -- @comments represents the tag 'copy':
326     -- copies a file or Fileset to a new file or directory
327     class Copy extends FileTask{
328         attribute file [0-1] : String;
329         attribute presserelastmodified [0-1] : String;
330         attribute tofile [0-1] : String;
331         attribute todir [0-1] : String;
332         attribute overwrite [0-1] : String;
333         attribute filtering [0-1] : String;
334         attribute flatten [0-1] : String;
335         attribute includeEmptyDirs [0-1] : String;
336         reference fileset [0-1] container : FileSet;
337         reference filterset [0-1] container : FilterSet;
338         reference mapper [0-1] container : Mapper;
339     }
340
341     -- @comments represents the tag 'delete':
342     -- deletes either a single file, all files and sub-directories
343     -- in a specified directory, or a set of files specified by one
344     -- or more FileSets
345     class Delete extends FileTask{
346         attribute file [0-1] : String;
347         attribute dir [0-1] : String;
348         attribute verbose [0-1] : String;
349         attribute quiet [0-1] : String;
350         attribute failonerror [0-1] : String;
351         attribute includeEmptyDirs [0-1] : String;
352         attribute includes [0-1] : String;
353         attribute includesfile [0-1] : String;
354         attribute excludes [0-1] : String;
355         attribute excludesfile [0-1] : String;
356         attribute defaultexcludes [0-1] : String;
357     }
358     -- @end fileTasks
359
360     -- @begin executionTasks
361     abstract class ExecutionTask extends PreDefinedTask{}
362
363     -- @comments represents the tag 'exec': executes a system command
364     class Exec extends ExecutionTask{
365         attribute executable : String;
366         attribute dir : String;
```

	ATL TRANSFORMATION EXAMPLE	
	Ant to Maven	Date 05/08/2005

```
367     }
368     -- @end executionTasks
369     -- @end antTasks
370     }
371     package PrimitiveTypes{
372         datatype String ;
373     }
```

III. XML2Ant.atl file

```
1  module XML2Ant;
2  create OUT : Ant from IN : XML;
3
4
5  -- -- to extract a list of String from a String -- --
6
7  -- helper getList: extract a sequence of String from the String listString -- in
8  the same order
9  -- (two elements are separated by a comma)
10 helper def:getList(listString: String):Sequence(String)=
11     if(listString.size()==0)
12         then Sequence{}
13         else thisModule.getListAux(listString,1,1,Sequence{})
14     endif;
15
16
17 -- helper getListAux
18 -- index1: begin of the word
19 -- index2: meter
20 helper def:getListAux(listString: String, index1: Integer, index2: Integer,
21 provSequence: Sequence(String)): Sequence(String)=
22     if (listString.size()<index2)
23         then provSequence -> append(listString.substring(index1,index2-1))
24     else
25         if (listString.substring(index2,index2)=' , ')
26             then thisModule.
27                 getListAux(listString,index2+1,index2+1, provSequence ->
28                     append(listString.substring(index1,index2-1)))
29         else thisModule.
30             getListAux(listString,index1,index2+1, provSequence)
31         endif
32     endif;
33
34
35 -- -- helper : to get an attribute -- --
36
37 -- helper getAttrVal: returns the value of the attribute 'name'
38 -- (without test of existence)
39 helper context XML!Element def:getAttrVal(name : String) : String =
40     self.children->
41         select(c | c.ocIsKindOf(XML!Attribute) and c.name = name)
42         ->first().value;
43
44 -- helper testAttribute: returns true if the attribute 'name' is defined
45 helper context XML!Element def: testAttribute(name : String) : Boolean =
46     not (self.children -> select(d | d.ocIsKindOf(XML!Attribute) and d.name = name)-
47 >
48         first().ocIsUndefined());
49
50
51 -- helper getAttribute: returns the value of the attribute given in
52 -- parameter
53 -- returns '' if this attribute does not exist
54 helper context XML!Element def:getAttribute(name : String):String =
55     if (self.testAttribute(name))
56         then self.getAttrVal(name)
57         else ''
58     endif;
```

```
59
60
61 -- -- others helpers -- --
62
63 -- helper testElement: returns true if the element 'name' is defined
64 helper context XML!Element def: testElement(name : String) : Boolean =
65     not (self.children ->
66         select(d | d.ocIsKindOf(XML!Element) and d.name = name)->
67             first().ocIsUndefined());
68
69 -- helper getText: returns the value of a text belonging to an element
70 -- 'name'
71 -- return '' if the element does not exist
72 helper context XML!Element def: getText(name : String) : String =
73     if self.testElement(name)
74     then self.children->
75         select(c | c.ocIsKindOf(XML!Element) and c.name=name) ->
76             first().children ->
77                 select(c | c.ocIsKindOf(XML!Text)) ->
78                     first().value
79     else ''
80     endif;
81
82
83 -- -- -- RULES -- -- --
84
85 -- central rule
86 rule Root2Project{
87     from i : XML!Root
88     to o : Ant!Project(
89         name <- i.getAttribute('name'),
90         basedir <- i.getAttribute('basedir'),
91         description <- i.getText('description'),
92         default <- XML!Element.allInstances() ->
93             select(d | d.name = 'target'
94                 and d.getAttribute('name')=i.getAttribute('default')) ->
95                 first(),
96         path <- i.children ->
97             select(d | d.ocIsKindOf(XML!Element) and d.name = 'path')->
98                 first(),
99         properties <- i.children ->
100             select(d | d.ocIsKindOf(XML!Element) and d.name = 'property'),
101         taskdef <- i.children ->
102             select(d | d.ocIsKindOf(XML!Element) and d.name = 'taskdef'),
103         targets <- i.children ->
104             select(d | d.ocIsKindOf(XML!Element) and d.name = 'target')
105     )
106 }
107
108 -- properties
109 rule PropertyLocation{
110     from i : XML!Element(
111         i.name = 'property' and
112         i.testAttribute('location')
113     )
114     to o : Ant!PropertyLocation(
115         name <- i.getAttribute('name'),
116         location <- i.getAttribute('location')
117     )
118 }
119
120 rule PropertyValue{
```

```
121     from i : XML!Element(
122         i.name = 'property' and
123         i.testAttribute('value')
124     )
125     to o : Ant!PropertyValue(
126         name <- i.getAttribute('name'),
127         value <- i.getAttribute('value')
128     )
129 }
130
131 rule PropertyFile{
132     from i : XML!Element(
133         i.name = 'property' and
134         i.testAttribute('file')
135     )
136     to o : Ant!PropertyFile(
137         file <- i.getAttribute('file')
138     )
139 }
140
141 rule PropertyEnv{
142     from i : XML!Element(
143         i.name = 'property' and
144         i.testAttribute('environment')
145     )
146     to o : Ant!PropertyEnv(
147         environment <- i.getAttribute('environment')
148     )
149 }
150
151
152 -- target
153 rule Target{
154     from i : XML!Element(
155         i.name = 'target'
156     )
157     to o : Ant!Target(
158         name <- i.getAttribute('name'),
159         description <- i.getAttribute('description'),
160         ifCondition <- i.getAttribute('if'),
161         unless <- i.getAttribute('unless'),
162         depends <- XML!Element.allInstances() ->
163             select(d | d.name = 'target'
164                 and thisModule.getList(i.getAttribute('depends'))->
165                     includes( d.getAttribute('name'))),
166         tasks <- i.children ->
167             select(d | d.oclIsKindOf(XML!Element))
168     )
169 }
170
171
172 -- tasks
173
174 -- concerning the taks defined by the user
175 -- definition of the task
176 rule TaskDef{
177     from i : XML!Element(
178         i.name = 'taskdef'
179     )
180     to o : Ant!TaskDef(
181         name <- i.getAttribute('name'),
182         classname <- i.getAttribute('classname')
```

```
183     )
184   }
185
186   -- call of a task created by the user
187   rule NewTask{
188     from i : XML!Element(
189       -- this task must be defined
190       not(XML!Element.allInstances() ->
191         select(d | d.name = 'taskdef'
192           and d.getAttribute('name')=i.name) ->
193           isEmpty())
194     )
195     to o : Ant!NewTask(
196       -- reference to the definition of this task
197       taskName <- XML!Element.allInstances() ->
198         select(d | d.name = 'taskdef'
199           and d.getAttribute('name')=i.name) ->
200           first(),
201       -- its attributes
202       attributes <- i.children ->
203         select(d | d.oclIsKindOf(XML!Attribute))
204     )
205   }
206
207   rule Attribut{
208     from i : XML!Attribute(
209       not(XML!Element.allInstances() ->
210         select(d | d.name = 'taskdef'
211           and d.getAttribute('name')=i.parent.name) ->
212           isEmpty())
213     )
214     to o : Ant!Attribut(
215       name <- i.name,
216       value<- i.value
217     )
218   }
219
220   -- pre-defined tasks
221
222   rule Mkdir{
223     from i : XML!Element(
224       i.name = 'mkdir'
225     )
226     to o : Ant!Mkdir(
227       dir <- i.getAttribute('dir')
228     )
229   }
230
231   rule Tstamp{
232     from i : XML!Element(
233       i.name = 'tstamp'
234     )
235     to o : Ant!Tstamp()
236   }
237
238   rule Java{
239     from i : XML!Element(
240       i.name = 'java'
241     )
242     to o : Ant!Java(
243       classname <- i.getAttribute('classname'),
244       jar <- i.getAttribute('jar'),
```

```
245     fork <- i.getAttribute('fork'),
246     classPath <- i.children ->
247     select(d | d.ocliIsKindOf(XML!Element) and d.name = 'classpath')
248   )
249 }
250
251 rule Javac{
252   from i : XML!Element(
253     i.name = 'javac'
254   )
255   to o : Ant!Javac(
256     destdir <- i.getAttribute('destdir'),
257     srcdir <- i.getAttribute('srcdir'),
258     classPath <- i.children ->
259     select(d | d.ocliIsKindOf(XML!Element) and d.name = 'classpath')->
260     first(),
261     inExcludes <- i.children ->
262     select(d | d.ocliIsKindOf(XML!Element) and
263       (d.name = 'include' or d.name = 'exclude'))
264   )
265 }
266
267 rule Javadoc{
268   from i : XML!Element(
269     i.name = 'javadoc'
270   )
271   to o : Ant!Javadoc(
272     sourcepath <- i.getAttribute('sourcepath'),
273     destdir <- i.getAttribute('destdir'),
274     packagenames <- i.getAttribute('packagenames'),
275     defaultexcludes <- i.getAttribute('defaultexcludes'),
276     author <- i.getAttribute('author'),
277     version <- i.getAttribute('version'),
278     use <- i.getAttribute('use'),
279     windowtitle <- i.getAttribute('windowtitle')
280   )
281 }
282
283 rule Copy{
284   from i : XML!Element(
285     i.name = 'copy'
286   )
287   to o : Ant!Copy(
288     todir <- i.getAttribute('todir'),
289     fileset <- i.children ->
290     select(d | d.ocliIsKindOf(XML!Element) and d.name = 'fileset') ->
291     first(),
292     filterset <- i.children ->
293     select(d | d.ocliIsKindOf(XML!Element) and d.name = 'filterset') ->
294     first()
295   )
296 }
297
298 rule Delete{
299   from i : XML!Element(
300     i.name = 'delete'
301   )
302   to o : Ant!Delete(
303     dir <- i.getAttribute('dir')
304   )
305 }
306
```



```
307 rule Jar{
308     from i : XML!Element(
309         i.name = 'jar'
310     )
311     to o : Ant!Jar(
312         jarfile <- i.getAttribute('jarfile'),
313         basedir <- i.getAttribute('basedir')
314     )
315 }
316
317 -- path, file and pattern
318 rule Path{
319     from i : XML!Element(
320         i.name = 'path'
321     )
322     to o : Ant!Path(
323         id <- i.getAttribute('id'),
324         refid <- i.getAttribute('refid'),
325         fileset <- i.children ->
326             select(d | d.ocliIsKindOf(XML!Element) and d.name = 'fileset')
327     )
328 }
329 rule FileSet{
330     from i : XML!Element(
331         i.name = 'fileset'
332     )
333     to o : Ant!FileSet(
334         dir <- i.getAttribute('dir'),
335         patternset <- i.children ->
336             select(d | d.ocliIsKindOf(XML!Element) and d.name = 'patternset'),
337         include <- i.children ->
338             select(d | d.ocliIsKindOf(XML!Element) and d.name = 'include'),
339         exclude <- i.children ->
340             select(d | d.ocliIsKindOf(XML!Element) and d.name = 'exclude')
341     )
342 }
343
344 rule PatternSet{
345     from i : XML!Element(
346         i.name = 'patternset'
347     )
348     to o : Ant!PatternSet(
349         inexcludes <- i.children ->
350             select(d | d.ocliIsKindOf(XML!Element) and
351                 (d.name = 'exclude' or d.name='include'))
352     )
353 }
354
355 rule ClassPath{
356     from i : XML!Element(
357         i.name = 'classpath'
358     )
359     to o : Ant!ClassPath(
360         refid <- i.getAttribute('refid'),
361         pathElement <- i.children ->
362             select(d | d.ocliIsKindOf(XML!Element) and d.name = 'pathElement'),
363         fileset <- i.children ->
364             select(d | d.ocliIsKindOf(XML!Element) and d.name = 'fileset')
365     )
366 }
367
368 rule PathElement{
```

```
369     from i : XML!Element(  
370         i.name = 'pathelement'  
371     )  
372     to o : Ant!PathElement(  
373         path <- i.getAttribute('path'),  
374         location <- i.getAttribute('location')  
375     )  
376 }  
377  
378 rule FilterSet{  
379     from i : XML!Element(  
380         i.name = 'filterset'  
381     )  
382     to o : Ant!FilterSet(  
383         starttoken <- i.getAttribute('starttoken'),  
384         endtoken <- i.getAttribute('endtoken'),  
385         filter <- i.children ->  
386             select(d | d.oclIsKindOf(XML!Element) and d.name = 'filter'),  
387         filtersfile <- i.children ->  
388             select(d | d.oclIsKindOf(XML!Element) and d.name = 'filtersfile')  
389     )  
390 }  
391  
392 rule Filter{  
393     from i : XML!Element(  
394         i.name = 'filter'  
395     )  
396     to o : Ant!Filter(  
397         token <- i.getAttribute('token'),  
398         value <- i.getAttribute('value')  
399     )  
400 }  
401  
402 rule FiltersFile{  
403     from i : XML!Element(  
404         i.name = 'filtersfile'  
405     )  
406     to o : Ant!FiltersFile(  
407         file <- i.getAttribute('file')  
408     )  
409 }  
410  
411 rule Includes{  
412     from i : XML!Element(  
413         i.name = 'include'  
414     )  
415     to o : Ant!Includes(  
416         name <- i.getAttribute('name'),  
417         ifCondition <- i.getAttribute('if'),  
418         unless <- i.getAttribute('unless')  
419     )  
420 }  
421  
422 rule Excludes{  
423     from i : XML!Element(  
424         i.name = 'exclude'  
425     )  
426     to o : Ant!Excludes(  
427         name <- i.getAttribute('name'),  
428         ifCondition <- i.getAttribute('if'),  
429         unless <- i.getAttribute('unless')  
430     )  
431 }
```

```
431 }
432
433 rule IncludesFile{
434   from i : XML!Element(
435     i.name = 'includesfile'
436   )
437   to o: Ant!IncludesFile(
438     name <- i.getAttribute('name'),
439     ifCondition <- i.getAttribute('if'),
440     unless <- i.getAttribute('unless')
441   )
442 }
443
444 rule ExcludesFile{
445   from i : XML!Element(
446     i.name = 'excludesfile'
447   )
448   to o : Ant!ExcludesFile(
449     name <- i.getAttribute('name'),
450     ifCondition <- i.getAttribute('if'),
451     unless <- i.getAttribute('unless')
452   )
453 }
```

IV. Ant2Maven.atl file

```
1  module Ant2Maven;
2  create OUTMaven : MavenMaven ,OUTProject : MavenProject from IN : Ant;
3
4  -- central element : Project
5  -- two files to create : MavenMaven (representing maven.xml)
6  --           and MavenProject (representing project.xml)
7  rule AntProject2Maven{
8      from a : Ant!Project(
9          if a.description.oclIsUndefined()
10             then false
11             else not (a.description='')
12             endif
13     )
14     -- for MavenProject
15     to mp : MavenProject!Project(
16         id <- a.name,
17         name <- a.name,
18         description <- a.description,
19         build <- mpBuild
20     ),
21     mpBuild : MavenProject!Build(
22         sourceDirectory <- a.basedir,
23         defaultGoal <- a.default.name
24     ),
25     -- for MavenMaven
26     mm : MavenMaven!Project(
27         xmlns <- itsXmlns,
28         default <- a.default,
29         path <- a.path,
30         properties <- a.properties,
31         taskdefs <- a.taskdef,
32         goals <- a.targets
33     ),
34     itsXmlns : MavenMaven!Xmlns(
35         name <- 'ant',
36         value <- 'jelly:ant'
37     )
38 }
39
40 rule AntProject2MavenWithoutDescription{
41     from a : Ant!Project(
42         if a.description.oclIsUndefined()
43             then true
44             else a.description=''
45             endif
46     )
47     -- for MavenProject
48     to mp : MavenProject!Project(
49         id <- a.name,
50         name <- a.name,
51         build <- mpBuild
52     ),
53     mpBuild : MavenProject!Build(
54         sourceDirectory <- a.basedir,
55         defaultGoal <- a.default.name
56     ),
57     -- for MavenMaven
58     mm : MavenMaven!Project(
```

```
59     xmlns <- itsXmlns,
60     default <- a.default,
61     path <- a.path,
62     properties <- a.properties,
63     taskdefs <- a.taskdef,
64     goals <- a.targets
65   ),
66   itsXmlns : MavenMaven!Xmlns(
67     name <- 'ant',
68     value <- 'jelly:ant'
69   )
70 }
71
72 -- rules only for Maven.xml (meta model : MavenMaven)
73
74 -- goals
75 rule AntTarget2MavenMavenGoal{
76   from a : Ant!Target
77   using {
78     itsDependencies : Sequence(Ant!Target) = a.depends->asSequence();
79   }
80   to mg : MavenMaven!Goal(
81     name <- a.name,
82     contentsGoal <- Sequence{dependencies,a.tasks}
83   ),
84   dependencies : distinct MavenMaven!AttainGoal
85     foreach(g in itsDependencies) (
86     attainGoal <- g
87   )
88 }
89
90 -- for the following rules : simple copy
91 -----
92 -- copy of Ant Properties
93
94 rule AntPropertyValue2MavenMavenAntPropertyValue{
95   from a : Ant!PropertyValue
96   to m : MavenMaven!AntPropertyValue(
97     name <- a.name,
98     value <- a.value
99   )
100 }
101
102 rule AntPropertyLocation2MavenMavenAntPropertyLocation{
103   from a : Ant!PropertyLocation
104   to m : MavenMaven!AntPropertyLocation(
105     name <- a.name,
106     location <- a.location
107   )
108 }
109
110
111 rule AntPropertyFile2MavenMavenAntPropertyFile{
112   from a : Ant!PropertyFile
113   to m : MavenMaven!AntPropertyFile(
114     file <- a.file)
115 }
116
117 rule AntPropertyEnv2MavenMavenAntPropertyEnv{
118   from a : Ant!PropertyEnv
119   to m : MavenMaven!AntPropertyEnv(
120     environment <- a.environment)
```



ATL TRANSFORMATION EXAMPLE

Ant to Maven

Date 05/08/2005

```
121 }
122
123 -- copy of tasks
124 -- java tasks
125 rule AntJava2MavenMavenJava{
126   from a : Ant!Java
127   to m : MavenMaven!Java(
128     classname <- a.classname,
129     jar <- a.jar,
130     fork <- a.fork,
131     classPath <- a.classPath
132   )
133 }
134
135 rule AntJavac2MavenMavenJavac{
136   from a : Ant!Javac
137   to m : MavenMaven!Javac(
138     destdir <- a.destdir,
139     srcdir <- a.srcdir,
140     classPath <- a.classPath,
141     inExcludes <- a.inExcludes
142   )
143 }
144
145 rule AntJavadoc2MavenMavenJavadoc{
146   from a : Ant!Javadoc
147   to m : MavenMaven!Javadoc(
148     sourcepath <- a.sourcepath,
149     destdir <- a.destdir,
150     packagenames <- a.packagenames,
151     defaultexcludes <- a.defaultexcludes,
152     author <- a.author,
153     version <- a.version,
154     use <- a.use,
155     windowtitle <- a.windowtitle
156   )
157 }
158
159 rule AntTstamp2MavenMavenTstamp{
160   from a : Ant!Tstamp
161   to m : MavenMaven!Tstamp()
162 }
163
164 rule AntJar2MavenMavenJar{
165   from a : Ant!Jar
166   to m : MavenMaven!Jar(
167     jarfile <- a.jarfile,
168     basedir <- a.basedir)
169 }
170
171 rule AntMkdir2MavenMavenMkdir{
172   from a : Ant!Mkdir
173   to m : MavenMaven!Mkdir(
174     dir <- a.dir)
175 }
176
177 rule AntCopy2MavenMavenCopy{
178   from a : Ant!Copy
179   to m : MavenMaven!Copy(
180     todir <- a.todir,
181     fileset <- a.fileset,
182     filterset <- a.filterset
```



ATL TRANSFORMATION EXAMPLE

Ant to Maven

Date 05/08/2005

```
183     )
184   }
185
186   rule AntDelete2MavenMavenDelete{
187     from a : Ant!Delete
188     to m : MavenMaven!Delete(
189       dir <- a.dir)
190   }
191
192   -- tasks defined by the user
193   rule AntTaskDef2MavenMavenTaskDef{
194     from a : Ant!TaskDef
195     to m : MavenMaven!AntTaskDef(
196       name <- a.name,
197       classname <- a.classname
198     )
199   }
200
201   rule AntNewTask2MavenMavenNewTask{
202     from a : Ant!NewTask
203     to m : MavenMaven!NewTask(
204       taskName <- a.taskName,
205       attributes <- a.attributes
206     )
207   }
208
209   rule AntAttribut2MavenMavenAttribut{
210     from a : Ant!Attribut
211     to m : MavenMaven!Attribut(
212       name <- a.name,
213       value <- a.value
214     )
215   }
216
217   -- copy for Path
218   rule AntPath2MavenMavenPath{
219     from a : Ant!Path
220     to mm : MavenMaven!Path(
221       id <- a.id,
222       refid <- a.refid,
223       fileset <- a.fileset,
224       path <- a.path,
225       pathElement <- a.pathElement
226     )
227   }
228
229   rule AntClassPath2MavenMavenClassPath{
230     from a : Ant!ClassPath
231     to mm : MavenMaven!ClassPath(
232       refid <- a.refid,
233       pathElement <- a.pathElement,
234       fileset <- a.fileset
235     )
236   }
237
238   rule AntPathElement2MavenMavenPathElement{
239     from a : Ant!PathElement
240     to mm : MavenMaven!PathElement(
241       path <- a.path,
242       location <- a.location
243     )
244   }
```

```
245
246 rule AntFileSet2MavenMavenFileSet{
247     from a : Ant!FileSet
248     to m : MavenMaven!FileSet(
249         dir <- a.dir,
250         patternset <- a.patternset,
251         include <- a.include,
252         exclude <- a.exclude
253     )
254 }
255
256 -- filters
257 rule AntFilterSet2MavenMavenFilterSet{
258     from a : Ant!FilterSet
259     to m : MavenMaven!FilterSet(
260         starttoken <- a.starttoken,
261         endtoken <- a.endtoken,
262         filter <- a.filter,
263         filtersfile <- a.filtersfile
264     )
265 }
266
267 rule AntFilter2MavenMavenFilter{
268     from a : Ant!Filter
269     to m : MavenMaven!Filter(
270         token <- a.token,
271         value <- a.value
272     )
273 }
274
275 rule AntFiltersFile2MavenMavenFiltersFile{
276     from a : Ant!FiltersFile
277     to m : MavenMaven!FiltersFile(
278         file <- a.file
279     )
280 }
281
282 -- pattern
283 rule AntPatternset2MavenMavenPatternset{
284     from a : Ant!PatternSet
285     to m : MavenMaven!PatternSet(
286         inexcludes <- a.inexcludes
287     )
288 }
289
290 rule AntIncludes2MavenMavenIncludes{
291     from a : Ant!Includes
292     to m : MavenMaven!Includes(
293         name <- a.name,
294         ifCondition <- a.ifCondition,
295         unless <- a.unless
296     )
297 }
298
299 rule AntExcludes2MavenMavenExcludes{
300     from a : Ant!Excludes
301     to m : MavenMaven!Excludes(
302         name <- a.name,
303         ifCondition <- a.ifCondition,
304         unless <- a.unless
305     )
306 }
```




**ATL
TRANSFORMATION EXAMPLE**

Ant to Maven

Date 05/08/2005

```
307
308 rule AntIncludesFile2MavenMavenIncludesFile{
309     from a : Ant!IncludesFile
310     to m : MavenMaven!IncludesFile(
311         name <- a.name,
312         ifCondition <- a.ifCondition,
313         unless <- a.unless
314     )
315 }
316
317 rule AntExcludesFile2MavenMavenExcludesFile{
318     from a : Ant!ExcludesFile
319     to m : MavenMaven!ExcludesFile(
320         name <- a.name,
321         ifCondition <- a.ifCondition,
322         unless <- a.unless
323     )
324 }
```

V. Maven2XML.atl file

```
1  module Maven2XML;
2  create XML1 : XMLMaven , XML2 : XMLProject
3      from InMaven : MavenMaven, InProject : MavenProject;
4
5  -- In this module, the two files are transformed in XML
6  -- but there is no link in the transformation
7
8  -- helper getXmlnsAux : returns the name of the Xmlns element having
9  -- the same value that given in parameter
10 helper def:getXmlnsAux(name: String): String =
11     MavenMaven!Xmlns.allInstances() ->
12     select(e|e.value=name)->first().name;
13
14 -- helper getXmlns : returns the prefix corresponding to name
15 helper def:getXmlns(name: String): String =
16     let completeValue: String = thisModule.getXmlnsAux(name) in
17     if completeValue.size()>0
18     then completeValue+':'
19     else ''
20     endif;
21
22 -- rules for the file representing maven.xml
23
24 -- central rule for maven.xml
25 rule MavenMavenProject2XMLMavenRoot{
26     from i : MavenMaven!Project
27     to o : XMLMaven!Root(
28         name <- 'project',
29         children <- Sequence {i.xmlns,goalDefault,
30                             i.path,i.properties,i.taskdefs,
31                             i.prePostGoals,i.goals}
32     ),
33     goalDefault : XMLMaven!Attribute (
34         name <- 'default',
35         value <- i.default.name
36     )
37 }
38
39
40
41 rule Xmlns{
42     from i : MavenMaven!Xmlns
43     to o:XMLMaven!Attribute(
44         name <- 'xmlns:'+i.name,
45         value <- i.value
46     )
47 }
48
49 -- Antproperty
50 rule PropertyValue{
51     from i : MavenMaven!AntPropertyValue
52     to o : XMLMaven!Element(
53         name <- thisModule.getXmlns('jelly:ant')+'property',
54         children <- Sequence{propertyName2,propertyValue}
55     ),
56     propertyName2 : XMLMaven!Attribute(
57         name <- 'name',
58         value <- i.name
```

```
59     ),
60     propertyValue : XMLMaven!Attribute(
61         name <- 'value',
62         value <- i.value
63     )
64 }
65 rule PropertyLocation{
66     from i : MavenMaven!AntPropertyLocation
67     to o : XMLMaven!Element(
68         name <- thisModule.getXmlns('jelly:ant')+ 'property',
69         children <- Sequence{propertyName2,propertyLocation}
70     ),
71     propertyName2 : XMLMaven!Attribute(
72         name <- 'name',
73         value <- i.name
74     ),
75     propertyLocation : XMLMaven!Attribute(
76         name <- 'location',
77         value <- i.location
78     )
79 }
80
81 rule PropertyFile{
82     from i : MavenMaven!AntPropertyFile
83     to o : XMLMaven!Element(
84         name <- thisModule.getXmlns('jelly:ant')+ 'property',
85         children <- nameFile
86     ),
87     nameFile : XMLMaven!Attribute(
88         name <- 'file',
89         value <- i.file
90     )
91 }
92
93 rule PropertyEnv{
94     from i : MavenMaven!AntPropertyEnv
95     to o : XMLMaven!Element(
96         name <- thisModule.getXmlns('jelly:ant')+ 'property',
97         children <- environmentName
98     ),
99     environmentName : XMLMaven!Attribute(
100         name <- 'environment',
101         value <- i.environment
102     )
103 }
104
105 -- goal
106 rule Goal{
107     from i : MavenMaven!Goal
108     to o : XMLMaven!Element(
109         name <- 'goal',
110         children <- Sequence{nameAttribute,i.contentsGoal}
111     ),
112     nameAttribute : XMLMaven!Attribute(
113         name <- 'name',
114         value <- i.name
115     )
116 }
117
118 rule AttainGoal{
119     from i : MavenMaven!AttainGoal
120     to o : XMLMaven!Element (
```

```
121     name <- 'attainGoal',
122     children <- attainGoalAttribute
123   ),
124   attainGoalAttribute : XMLMaven!Attribute (
125     name <- 'name',
126     value <- i.attainGoal.name
127   )
128 }
129
130 rule PreGoal{
131   from i : MavenMaven!PreGoal
132   to o : XMLMaven!Element(
133     name <- 'preGoal',
134     children <- Sequence{nameAttribute,i.contentsGoal}
135   ),
136   nameAttribute : XMLMaven!Attribute(
137     name <- 'name',
138     value <- i.centralGoal.name
139   )
140 }
141
142 rule PostGoal{
143   from i : MavenMaven!PostGoal
144   to o : XMLMaven!Element(
145     name <- 'postGoal',
146     children <- Sequence{nameAttribute,i.contentsGoal}
147   ),
148   nameAttribute : XMLMaven!Attribute(
149     name <- 'name',
150     value <- i.centralGoal.name
151   )
152 }
153
154 -- jelly commands
155 rule JellySet{
156   from i:MavenMaven!JellySet
157   to o : XMLMaven!Element(
158     name <- thisModule.getXmlns('jelly:core')+'set',
159     children <- Sequence{varAttribute,valueAttribute}
160   ),
161   varAttribute : XMLMaven!Attribute(
162     name <- 'var',
163     value <- i.var
164   ),
165   valueAttribute : XMLMaven!Attribute(
166     name <- 'value',
167     value <- i.value
168   )
169 }
170
171 -- tasks
172 -- task defined by the user
173 rule TaskDef{
174   from i : MavenMaven!AntTaskDef
175   to o : XMLMaven!Element(
176     name <- thisModule.getXmlns('jelly:ant')+'taskdef',
177     children <- Sequence{nameName,nameClassName}
178   ),
179   nameName : XMLMaven!Attribute(
180     name <- 'name',
181     value <- i.name
182   ),
```

```
183     nameClassName : XMLMaven!Attribute(  
184         name <- 'classname',  
185         value <- i.classname  
186     )  
187 }  
188  
189 rule NewTask{  
190     from i : MavenMaven!NewTask  
191     to o : XMLMaven!Element(  
192         name <- i.taskName.name,  
193         children <- i.attributes  
194     )  
195 }  
196  
197 rule Attribut{  
198     from i : MavenMaven!Attribut  
199     to o : XMLMaven!Attribute(  
200         name <- i.name,  
201         value <- i.value  
202     )  
203 }  
204  
205 -- pre defined tasks  
206 rule Tstamp{  
207     from i : MavenMaven!Tstamp  
208     to o : XMLMaven!Element(  
209         name <- thisModule.getXmlns('jelly:ant')+ 'tstamp'  
210     )  
211 }  
212  
213 rule Mkdir{  
214     from i : MavenMaven!Mkdir  
215     to o : XMLMaven!Element(  
216         name <- thisModule.getXmlns('jelly:ant')+ 'mkdir',  
217         children <- dirAttribute  
218     ),  
219     dirAttribute : XMLMaven!Attribute(  
220         name <- thisModule.getXmlns('jelly:ant')+ 'dir',  
221         value <- i.dir  
222     )  
223 }  
224  
225 rule Javac{  
226     from i : MavenMaven!Javac  
227     to o : XMLMaven!Element(  
228         name <- thisModule.getXmlns('jelly:ant')+ 'javac',  
229         children <- Sequence{sourceDirAttribute, destDirAttribute,  
230                                 i.inExcludes, i.classPath}  
231     ),  
232     sourceDirAttribute : XMLMaven!Attribute(  
233         name <- 'srcdir',  
234         value <- i.srcdir  
235     ),  
236     destDirAttribute : XMLMaven!Attribute(  
237         name <- 'destdir',  
238         value <- i.destdir  
239     )  
240 }  
241  
242 rule Copy{  
243     from i : MavenMaven!Copy  
244     to o : XMLMaven!Element(  


```

```
245     name <- thisModule.getXmlns('jelly:ant')+ 'copy',
246     children <- Sequence{toDirAttribute,i.fileset}
247   },
248   toDirAttribute : XMLMaven!Attribute(
249     name <- 'todir',
250     value <- i.todir
251   )
252 }
253
254
255
256 -----
257 -----
258 -- path,pattern and filter (like ANT2XML)
259 rule Path{
260   from i : MavenMaven!Path
261   to o : XMLMaven!Element(
262     name <- thisModule.getXmlns('jelly:ant')+ 'path',
263     children <- Sequence{idAttribute,i.fileset,i.path,i.pathElement}
264   ),
265   idAttribute : XMLMaven!Attribute(
266     name <- 'id',
267     value <- i.id
268   )
269 }
270
271 rule ClassPath{
272   from i : MavenMaven!ClassPath
273   to o : XMLMaven!Element(
274     name <- thisModule.getXmlns('jelly:ant')+ 'classpath',
275     children <- refidAttribute),
276   refidAttribute : XMLMaven!Attribute(
277     name <- 'refid',
278     value <- i.refid
279   )
280 }
281
282 rule Fileset{
283   from i : MavenMaven!FileSet
284   to o : XMLMaven!Element(
285     name <- thisModule.getXmlns('jelly:ant')+ 'fileset',
286     children <- Sequence{dirAttribute,i.patternset,i.include,i.exclude}
287   ),
288   dirAttribute : XMLMaven!Attribute(
289     name <- 'dir',
290     value <- i.dir
291   )
292 }
293
294 rule PathElement{
295   from i : MavenMaven!PathElement
296   to o : XMLMaven!Element(
297     name <- thisModule.getXmlns('jelly:ant')+ 'pathelement'
298   )
299 }
300
301 rule PatternSet{
302   from i : MavenMaven!PatternSet
303   to o : XMLMaven!Element(
304     name <- thisModule.getXmlns('jelly:ant')+ 'patternset',
305     children <- i.inexcludes
306   )

```

```
307 }
308
309 rule Include{
310   from i : MavenMaven!Includes
311   to o : XMLMaven!Element(
312     name <- thisModule.getXmlns('jelly:ant')+'include',
313     children <- nameAttribute
314   ),
315   nameAttribute : XMLMaven!Attribute(
316     name <- 'name',
317     value <- i.name
318   )
319 }
320
321 rule Exclude{
322   from i : MavenMaven!Excludes
323   to o : XMLMaven!Element(
324     name <- thisModule.getXmlns('jelly:ant')+'exclude',
325     children <- nameAttribute
326   ),
327   nameAttribute : XMLMaven!Attribute(
328     name <- 'name',
329     value <- i.name
330   )
331 }
332
333
334 -- rules for the file representing project.xml
335 -- central rule for project.xml
336 rule MavenProjectProject2XMLProjectRoot{
337   from i : MavenProject!Project(
338     if i.description.oclIsUndefined()
339     then false
340     else not (i.description='')
341     endif
342   )
343   to o : XMLProject!Root(
344     name <- 'project',
345     children <- Sequence{idAttribute,nameAttribute,
346                          descriptionElement,i.build}
347   ),
348   idAttribute : XMLProject!Attribute(
349     name <- 'id',
350     value <- i.id
351   ),
352   nameAttribute : XMLProject!Attribute(
353     name <- 'name',
354     value <- i.name
355   ),
356   descriptionElement : XMLProject!Element(
357     name <- 'description',
358     children <- descriptionText
359   ),
360   descriptionText : XMLProject!Text(
361     value <- i.description
362   )
363 }
364
365 rule MavenProjectProject2XMLProjectRootWithoutDescription{
366   from i : MavenProject!Project(
367     if i.description.oclIsUndefined()
368     then true
```

```
369         else i.description=''
370         endif
371     )
372     to o : XMLProject!Root(
373         name <- 'project',
374         children <- Sequence{idAttribute,nameAttribute,i.build}
375     ),
376     idAttribute : XMLProject!Attribute(
377         name <- 'id',
378         value <- i.id
379     ),
380     nameAttribute : XMLProject!Attribute(
381         name <- 'name',
382         value <- i.name
383     )
384 }
385
386
387 rule MavenProjectDescription{
388     from i : MavenProject!Description
389     to o : XMLProject!Element(
390         name <- 'description',
391         children <- textText
392     ),
393     textText : XMLProject!Text(
394         value <- i.description
395     )
396 }
397
398 rule MavenProjectBuild{
399     from i : MavenProject!Build
400     to o : XMLProject!Element(
401         name <- 'build',
402         children <- Sequence{defaultGoalElement,sourceDirectoryElement}
403     ),
404     defaultGoalElement : XMLProject!Element(
405         name <- 'defaultGoal',
406         children <- defaultGoalText
407     ),
408     defaultGoalText : XMLProject!Text(
409         value <- i.defaultGoal
410     ),
411     sourceDirectoryElement : XMLProject!Element(
412         name <- 'sourceDirectory',
413         children <- sourceDirectoryText
414     ),
415     sourceDirectoryText : XMLProject!Text(
416         value <- i.sourceDirectory
417     )
418 }
```


	ATL TRANSFORMATION EXAMPLE	
	Ant to Maven	Date 05/08/2005

References

- [1] Ant Overview. <http://ant.apache.org/manual/>
- [2] Maven Overview. <http://maven.apache.org/reference/project-descriptor.html>
- [3] KM3: Kernel MetaMetaModel. <http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/gmt-home/doc/atl/index.html>.